## Course Project: Understanding paths to Software Performance in the Future

**Assigned:** October 23, 2013
**Due:** December 3, 2013 In-class. Turn in to Professor Chien or the TA's.

With the challenges that computers are facing due to Moore's Law scaling, computer architecture is pursuing two major directions to continue performance increase. In this project, you will explore the impact these trends have on software design, effort, and what will be required to achieve good performance (and energy efficiency in the future).

**The Programs:**

You are being given two programs, matrix-matrix multiply, and a sorting program. They are available in the project-programs.zip file. They will be used as two exemplars for the much more complex large-scale computing problems we will face in the future. You should select one of the two options below, and at the end of the course, we will share results and discuss the prospects for software performance in the future (and the effort required to achieve it).

**Option A: Accelerating Programs on a single core (and thread)**

Your objective is to make the two programs run as fast as possible on a single thread of a single core. You will report performance relative to a baseline of simple compilation of the supplied source code. You may use the CSIL linux machines for this work, or any x86 machine of your choosing, but be sure to make all of your measurements on the same machine, and document the hardware configuration, compiler switches, and program modifications made, along with the performance of course.

You may try any of the following things:

- Compilation with different compilers
- Compilation with different optimization switches (but make sure not to use the switches that make use of multiple threads)
- Looking at the assembly code ("-S")
- Rewriting the program for greater efficiency, including use of SSE intrinsics
- Rewriting with a different algorithm or data structures
- Rewriting for data locality

**Option B: Accelerating Programs on a multi-core processor**

Your objective is to make the two programs run as fast as possible on a multi-thread, multi-core system. You will report performance relative to a baseline simple compilation of the supplied source code. You may use the CSIL linux machines for this work, or any x86 machine of your choosing, but be sure to make all of your measurements on the same machine, and document the hardware configuration, compiler switches, and program modifications made, along with the performance of course.

- Compilation with different compilers
- Compilation with different optimization switches (yes, you can use the ones for automatic parallelization using threads)
- Rewriting the program for greater efficiency, including using OpenMP commands

- Rewriting with a different algorithm or data structures
- Rewriting for data locality

In this case, you should take data for 1, 2, 3, 4, .... threads up to the maximum number of threads the machine supports.

**Overall Comments**

You may choose whatever size inputs you would like for the programs, and measure relative to a run of the unoptimized program on the same input.

You should design your experiments to focus on measurement of the "work part" of the program. Either the sort of matrixmult, not the setup or checks.

As a little background information, the CSIL machines have the following C compilers (that we found): gcc, g++, CLANG. All of these compilers support both SSE and OpenMP. See their "man" pages, but also the following links for more information:

SSE
http://gcc.gnu.org/onlinedocs/gcc-3.1/gcc/X86-Built-in-Functions.html
http://www.linuxjournal.com/content/introduction-gcc-compiler-intrinsics-vector-processing
http://software.intel.com/en-us/articles/intel-intrinsics-guide

OpenMP
http://gcc.gnu.org/wiki/openmp
http://openmp.org/wp/

**What you should turn in:**

We expect a writeup of 5-10 pages (including graphs and tables!) answering the questions below. In addition, you should submit the final version of your code for each of the two benchmarks.

1. Which option you chose
2. A description of the sequence of steps you tried, and the performance achieved for each step (be clear, and choose your test sizes appropriately). Code snippets in the text as helpful.
3. Corresponding source code for the significant versions (versions and names, we'll setup a way to submit this)
4. What an explanation of WHY the performance changed at each step (might require looking at the assembly code)
5. The final performance you achieved, and how much faster it is than the original implementation
6. For those who did Option A: how much more might performance be increased for each of these application through the extension of vectors to longer lengths?
7. For those who did Option B: how much more might performance be increase for each of these applications through the extension to more threads and cores? (extrapolate based on the data you were able to collect)

All of the teaching staff is happy to engage in discussions about this as we go along, and discussions in the Piazza forum are welcome as well. However, the discussion should focus on the tools, and ideas for how to optimize programs. We expect each student to independently perform a set of performance experiments, and independently understand/analyze those experiments.